
NLTK-Trainer Documentation

Release 1.0

Jacob Perkins

Aug 02, 2018

Contents

1	Download	3
2	Documentation	5
2.1	Training Classifiers	5
2.2	Using a Trained Classifier	6
2.3	Training Part of Speech Taggers	7
2.4	Using a Trained Tagger	7
2.5	Training IOB Chunkers	8
2.6	Using a Trained Chunker	8
2.7	Analyzing a Tagged Corpus	9
2.8	Analyzing Tagger Coverage	9
3	Books	11
4	Articles	13
5	Demos and APIs	15
6	Indices and tables	17

NLTK-Trainer is a set of [Python](#) command line scripts for natural language processing. With these scripts, you can do the following things without writing a single line of code:

1. train [NLTK](#) based models
2. evaluate pickled models against a corpus
3. analyze a corpus

These scripts are Python 2 & 3 compatible and work with NLTK 2.0.4 and higher.

CHAPTER 1

Download

The scripts can be downloaded from [nltk-trainer](#) on github.

2.1 Training Classifiers

Example usage with the `movie_reviews` corpus can be found in [Training Binary Text Classifiers with NLTK Trainer](#).

Train a binary NaiveBayes classifier on the `movie_reviews` corpus, using paragraphs as the training instances:

```
python train_classifier.py movie_reviews --instances paras --classifier NaiveBayes
```

Include bigrams as features: `python train_classifier.py movie_reviews --instances paras --classifier NaiveBayes --ngrams 1 --ngrams 2`

Minimum score threshold: `python train_classifier.py movie_reviews --instances paras --classifier NaiveBayes --ngrams 1 --ngrams 2 --min_score 3`

Maximum number of features: `python train_classifier.py movie_reviews --instances paras --classifier NaiveBayes --ngrams 1 --ngrams 2 --max_feats 1000`

Use the default Maxent algorithm: `python train_classifier.py movie_reviews --instances paras --classifier Maxent`

Use the MEGAM Maxent algorithm: `python train_classifier.py movie_reviews --instances paras --classifier MEGAM`

Train on files instead of paragraphs: `python train_classifier.py movie_reviews --instances files --classifier MEGAM`

Train on sentences: `python train_classifier.py movie_reviews --instances sents --classifier MEGAM`

Evaluate the classifier by training on 3/4 of the paragraphs and testing against the remaining 1/4, without pickling:

```
python train_classifier.py movie_reviews --instances paras --classifier NaiveBayes --fraction 0.75 --no-pickle
```

The following classifiers are available:

- NaiveBayes

- `DecisionTree`
- `Maxent` with various algorithms (many of these require `numpy` and `scipy`, and MEGAM requires `megam`)
- `Svm` (requires `svmlight` and `pysvmlight`)

If you also have `scikit-learn` then the following classifiers will also be available, with `sklearn` specific training options. If there is a `sklearn` classifier or training option you want that is not present, please [submit an issue](#).

- `sklearn.ExtraTreesClassifier`
- `sklearn.GradientBoostingClassifier`
- `sklearn.RandomForestClassifier`
- `sklearn.LogisticRegression`
- `sklearn.BernoulliNB`
- `sklearn.GaussianNB`
- `sklearn.MultinomialNB`
- `sklearn.KNeighborsClassifier`
- `sklearn.LinearSVC`
- `sklearn.NuSVC`
- `sklearn.SVC`
- `sklearn.DecisionTreeClassifier`

For example, here's how to use the `sklearn.LinearSVC` classifier with the `movie_reviews` corpus:

```
python train_classifier.py movie_reviews --classifier sklearn.LinearSVC
```

For a complete list of usage options: `python train_classifier.py --help`

There are also many usage examples shown in Chapter 7 of [Python 3 Text Processing with NLTK 3 Cookbook](#).

2.2 Using a Trained Classifier

You can use a trained classifier by loading the pickle file using `nltk.data.load`:

```
>>> import nltk.data
>>> classifier = nltk.data.load("classifiers/NAME_OF_CLASSIFIER.pickle")
```

Or if your classifier pickle file is not in a `nltk_data` subdirectory, you can load it with `pickle.load`:

```
>>> import pickle
>>> classifier = pickle.load(open("/path/to/NAME_OF_CLASSIFIER.pickle"))
```

Either method will return an object that supports the `ClassifierI` interface.

Once you have a `classifier` object, you can use it to classify word features with the `classifier.classify(feats)` method.

```
>>> words = ['some', 'words', 'in', 'a', 'sentence']
>>> feats = dict([(word, True) for word in words])
>>> classifier.classify(feats)
```

If you used the `--ngrams` option with values greater than 1, you should include these ngrams in the dictionary using `nltk.util.ngrams`.

```
>>> from nltk.util import ngrams
>>> words = ['some', 'words', 'in', 'a', 'sentence']
>>> feats = dict([(word, True) for word in words + ngrams(words, n)])
>>> classifier.classify(feats)
```

The list of words you use for creating the feature dictionary should be created by [tokenizing](#) the appropriate text instances: sentences, paragraphs, or files depending on the `--instances` option.

Most of the sentiment classifiers used by [text-processing.com](#) were trained with `train_classifier.py`.

2.3 Training Part of Speech Taggers

The `train_tagger.py` script can use any corpus included with NLTK that implements a `tagged_sents()` method. It can also train on the `timit` corpus, which includes tagged sentences that are not available through the `TimitCorpusReader`.

Example usage can be found in [Training Part of Speech Taggers with NLTK Trainer](#).

Train the default sequential backoff tagger on the treebank corpus: `python train_tagger.py treebank`

To use a brill tagger with the default initial tagger: `python train_tagger.py treebank --brill`

To train a NaiveBayes classifier based tagger, without a sequential backoff tagger: `python train_tagger.py treebank --sequential '' --classifier NaiveBayes`

To train a unigram tagger: `python train_tagger.py treebank --sequential u`

To train on the switchboard corpus: `python train_tagger.py switchboard`

To train on a custom corpus, whose fileids end in “.pos”, using a [TaggedCorpusReader](#): `python train_tagger.py /path/to/corpus --reader nltk.corpus.reader.tagged.TaggedCorpusReader --fileids '.*\.pos'`

The corpus path can be absolute, or relative to a `nltk_data` directory. For example, both `corpora/treebank/tagged` and `/usr/share/nltk_data/corpora/treebank/tagged` will work.

You can also restrict the files used with the `--fileids` option: `python train_tagger.py conll2000 --fileids train.txt`

For a complete list of usage options: `python train_tagger.py --help`

There are also many usage examples shown in Chapter 4 of [Python 3 Text Processing with NLTK 3 Cookbook](#).

2.4 Using a Trained Tagger

You can use a trained tagger by loading the pickle file using `nltk.data.load`:

```
>>> import nltk.data
>>> tagger = nltk.data.load("taggers/NAME_OF_TAGGER.pickle")
```

Or if your tagger pickle file is not in a `nltk_data` subdirectory, you can load it with `pickle.load`:

```
>>> import pickle
>>> tagger = pickle.load(open("/path/to/NAME_OF_TAGGER.pickle"))
```

Either method will return an object that supports the `TaggerI` interface.

Once you have a `tagger` object, you can use it to tag sentences (or lists of words) with the `tagger.tag(words)` method:

```
>>> tagger.tag(['some', 'words', 'in', 'a', 'sentence'])
```

`tagger.tag(words)` will return a list of 2-tuples of the form `[(word, tag)]`.

All of the taggers demonstrated at text-processing.com were trained with `train_tagger.py`.

2.5 Training IOB Chunkers

The `train_chunker.py` script can use any corpus included with NLTK that implements a `chunked_sents()` method.

Train the default sequential backoff tagger based chunker on the `treebank_chunk` corpus:: `python train_chunker.py treebank_chunk`

To train a NaiveBayes classifier based chunker: `python train_chunker.py treebank_chunk --classifier NaiveBayes`

To train on the `conll2000` corpus: `python train_chunker.py conll2000`

To train on a custom corpus, whose fileids end in “.pos”, using a `ChunkedCorpusReader`: `python train_chunker.py /path/to/corpus --reader nltk.corpus.reader.chunked.ChunkedCorpusReader --fileids '.*\.pos'`

The corpus path can be absolute, or relative to a `nltk_data` directory. For example, both `corpora/treebank/tagged` and `/usr/share/nltk_data/corpora/treebank/tagged` will work.

You can also restrict the files used with the `--fileids` option: `python train_chunker.py conll2000 --fileids train.txt`

For a complete list of usage options: `python train_chunker.py --help`

There are also many usage examples shown in Chapter 5 of *Python 3 Text Processing with NLTK 3 Cookbook*.

2.6 Using a Trained Chunker

You can use a trained chunker by loading the pickle file using `nltk.data.load`:

```
>>> import nltk.data
>>> tagger = nltk.data.load("chunkers/NAME_OF_CHUNKER.pickle")
```

Or if your chunker pickle file is not in a `nltk_data` subdirectory, you can load it with `pickle.load`:

```
>>> import pickle
>>> tagger = pickle.load(open("/path/to/NAME_OF_CHUNKER.pickle"))
```

Either method will return an object that supports the `ChunkerParserI` interface. But before you can use this chunker, you must

```
>>> chunker.parse(tagged_words)
```

`chunker.parse(tagged_words)` will return a `Tree` whose subtrees will be chunks, and whose leaves are the original tagged words.

All of the chunkers demonstrated at text-processing.com were trained with `train_chunker.py`.

2.7 Analyzing a Tagged Corpus

The `analyze_tagged_corpus.py` script will show the following statistics about a tagged corpus:

- total number of words
- number of unique words
- number of tags
- the number of times each tag occurs

Example output can be found in [Analyzing Tagged Corpora and NLTK Part of Speech Taggers](#).

To analyze the treebank corpus: `python analyze_tagged_corpus.py treebank`

To sort the output by tag count from highest to lowest: `python analyze_tagged_corpus.py treebank --sort count --reverse`

To see simplified tags, instead of standard tags: `python analyze_tagged_corpus.py treebank --simplify_tags`

To analyze a custom corpus, whose fileids end in “.pos”, using a `TaggedCorpusReader`: `python analyze_tagged_corpus.py /path/to/corpus --reader nltk.corpus.reader.tagged.TaggedCorpusReader --fileids '.+\pos'`

The corpus path can be absolute, or relative to a `nltk_data` directory. For example, both `corpora/treebank/tagged` and `/usr/share/nltk_data/corpora/treebank/tagged` will work.

For a complete list of usage options: `python analyze_tagged_corpus.py --help`

2.8 Analyzing Tagger Coverage

The `analyze_tagger_coverage.py` script will run a part-of-speech tagger over a corpus to determine how many times each tag is found. Example output can be found in [Analyzing Tagged Corpora and NLTK Part of Speech Taggers](#).

Here’s an example using the NLTK default tagger on the treebank corpus: `python analyze_tagger_coverage.py treebank`

To get detailed metrics on each tag, you can use the `--metrics` option. This requires using a tagged corpus in order to compare actual tags against tags found by the tagger. See [NLTK Default Tagger Treebank Tag Coverage](#) and [NLTK Default Tagger CoNLL2000 Tag Coverage](#) for examples and statistics.

The default tagger used is NLTK’s default tagger. To analyze the coverage using a different tagger, use the `--tagger` option with `python analyze_tagger_coverage.py treebank --tagger /path/to/tagger.pickle`

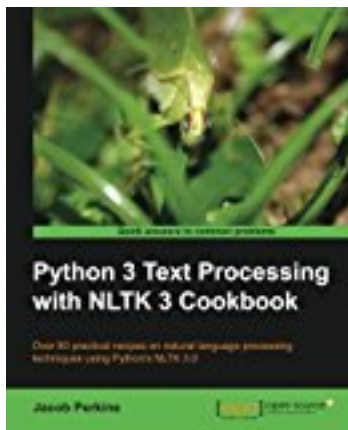
You can also analyze tagger coverage over a custom corpus. For example, with a corpus whose fileids end in “.pos”, you can use `python analyze_tagger_coverage.py /path/to/corpus --reader nltk.corpus.reader.tagged.TaggedCorpusReader --fileids '.+\pos'`

The corpus path can be absolute, or relative to a `nltk_data` directory. For example, both `corpora/treebank/tagged` and `/usr/share/nltk_data/corpora/treebank/tagged` will work.

For a complete list of usage options: `python analyze_tagger_coverage.py --help`

CHAPTER 3

Books



Python 3 Text Processing with NLTK 3 Cookbook contains many examples for training NLTK models with & without NLTK-Trainer.

- Chapter 4 covers part-of-speech tagging and *train_tagger.py*.
- Chapter 5 shows how to train phrase chunkers and use *train_chunker.py*.
- Chapter 7 demonstrates classifier training and *train_classifier.py*.

CHAPTER 4

Articles

- [Training Binary Classifiers with NLTK Trainer](#)
- [Training Part of Speech Taggers with NLTK Trainer](#)
- [Analyzing Tagger Corpora and NLTK Part of Speech Taggers](#)
- [NLTK Default Tagger Coverage of treebank corpus](#)
- [NLTK Default Tagger Coverage of conll2000 corpus](#)

CHAPTER 5

Demos and APIs

Nearly all the models that power the text-processing.com NLTK demos and NLP APIs have been trained using NLTK-Trainer.

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`